Fast 3-D prestack depth migration with a parallel PSPI algorithm

Peter M. Roberts, Douglas M. Alde, Leigh S. House, Los Alamos National Laboratory; Joseph H. Higginbotham, Dwight V. Sukup, Texaco Exploration and Production Technology Department*

Summary

A parallel computing model is presented that produces scaleable code for performing 3D prestack depth migration using the phase-shift-plus-interpolation (PSPI) algorithm. The parallel code is constructed by building standard PVM message-passing calls into an existing serial code. Very few modifications are required to the original algorithm because the parallel model simply farms out individual shot records to a number of machine processing elements (PE's) that are each running an independently spawned copy of the migration code. This simple approach to parallelism allows large 3D seismic surveys to be migrated in a fraction of the time normally required by the equivalent serial code on a single-PE computer. The PVM model can be used to port serial codes to different types of parallel machines with minimal development effort. A description of the parallel model is shown, along with benchmark performance results for code versions written for a workstation cluster and a Cray T3D, using a subset of a full 3D seismic survey as test data. A factor of 100 increase in effective processing speed was observed for a 128-PE Cray T3D run, relative to a serial single-PE run on a Sun SPARC workstation.

Introduction

As domestic oil and gas reserves become harder to locate, the need for more accurate seismic imaging of reservoirs increases. High-resolution imaging in geologically complex fields requires 3-D prestack depth migration using data collected from very large seismic surveys. The PSPI algorithm [Gazdag & Sguazzero, 1984] is considered to be one of the more accurate, stabile and efficient methods for migrating seismic data collected in regions with strong lateral heterogeneity. Discussions of the most popular migration methods, including PSPI, can be found in [Yilmaz, 1987]. PSPI involves free propagation of the recorded wavefield in the wavenumber domain followed by interpolation of the intermediate results for different reference velocities in the spatial domain. This sequence is repeated iteratively as the wavefield propagates downward through the initial velocity model. The efficiency of the algorithm arises through the use of repeated Fast Fourier Transforms (FFT) between the space and wavenumber domains. Usually PSPI is performed on one common-shot or common-receiver record at a time, and the migrated results are superposed into a fixed-size image volume. Typical types of 3-D seismic surveys that can provide common-shot or common-receiver data include land surveys, vertical cable acquisition, and multiple-streamer schemes. Because the migrated results of one shot record do not depend on other records, large 3-D surveys can be broken up into small enough pieces that will fit within almost any computer's available memory. Thus, the limiting factor for processing large 3-D surveys is the computing time required, and single computers are not fast enough to allow for economically feasible exploration using serial computing models.

We have converted an existing serial PSPI code into a parallel-processor version that runs on both heterogeneous workstation clusters and on the Cray T3D massively parallel processor (MPP) machine. The parallel code employs a master/slave message-passing model implemented under the Parallel Virtual Machine (PVM) programming environment. By farming out individual common-shot records, the processing speed for a complete 3D data set scales approximately with the number of processing elements in the virtual machine. This means that migration runs that normally would require, say, 3 months of computing could be completed in 1 day or less. We describe here the structure of the PVM implementation used, compare it with the original serial code structure and give performance benchmark results for several different computing platforms. The parallel model requires very little modification of the original PSPI code and, because no communication is required between the slave processes, the model can be applied, in theory, to other migration algorithms that can process individual shot records separately.

Parallel Model Design

PSPI is a natural candidate for message-passing parallelism because of the inherent independence of common-shot records. Although elegant schemes can be devised which exploit the data-parallel features of frequency-wavenumber data, these approaches are usually difficult to program, result in machine-dependent codes, and in the end do not always provide the expected increase in performance. A much simpler approach is to spawn multiple copies of the same code and have each copy work on its own subset of shot records. As each record is migrated, the results are added into a common image volume that each spawned process has access to. The only communication required among PE's is information about which records to process, where the migrated results belong in the image volume, and a signal indicating when the volume is currently being updated by another PE. This type of communication, or message-passing, is best implemented using the so-called master/slave model. Of the several types of master/slave models that have been proposed for parallel computing, the one we use is the "crowd" or "fan-in-fan-out" (FIFO) approach. The FIFO implementation uses a single master process that determines how many machine PE's are available and spawns a single slave migration process on each PE. The master also tells each slave which shot record to process next, receives the migrated data from each slave, and takes charge of updating the image volume. When all records have been processed, the master terminates the job. The details of the PVM implementation are best described by comparisons with the original serial structure.

Figure 1 shows the control structure for the original serial code and Figure 2 shows the changes that were made for the parallel PVM version. In the serial code, the main program reads migration parameters and database information from disk files and determines what records will be processed and where the migrated data will fit in the image volume. Subroutines are called that read in data for the next shot record and FFT them into the frequency domain. Another subroutine reads information about the velocity model and generates velocity data to be used in the migration. Finally, the PSPI algorithm itself is contained in another set of subroutines that perform the downward continuation and imaging steps of the migration. Migrated data for each depth step are passed back to the main program and these are added into the image volume and saved to disk. Due to memory limitations, the velocity and raw FFT data are held in temporary disk files and accessed as needed by the main program. Although, in theory, these temporary files can still be used in a parallel code, the frequent disk I/O would cause severe bottlenecks unless each PE was reading and writing to its own local disk. This is often not the case, and, in particular, would cause prohibitive I/O bottlenecks on the Cray T3D. Because our eventual goal was to develop an efficient T3D version, our initial strategy was to write a version for a workstation cluster that does not use the scratch files. This was achieved by carefully sizing array dimensions in the serial code to allow the velocities and non-zero FFT data to fit completely in machine memory. The remaining changes, then, were specifically designed to replace most of the main program's control functions with a separate master program that controls multiple slave processes.

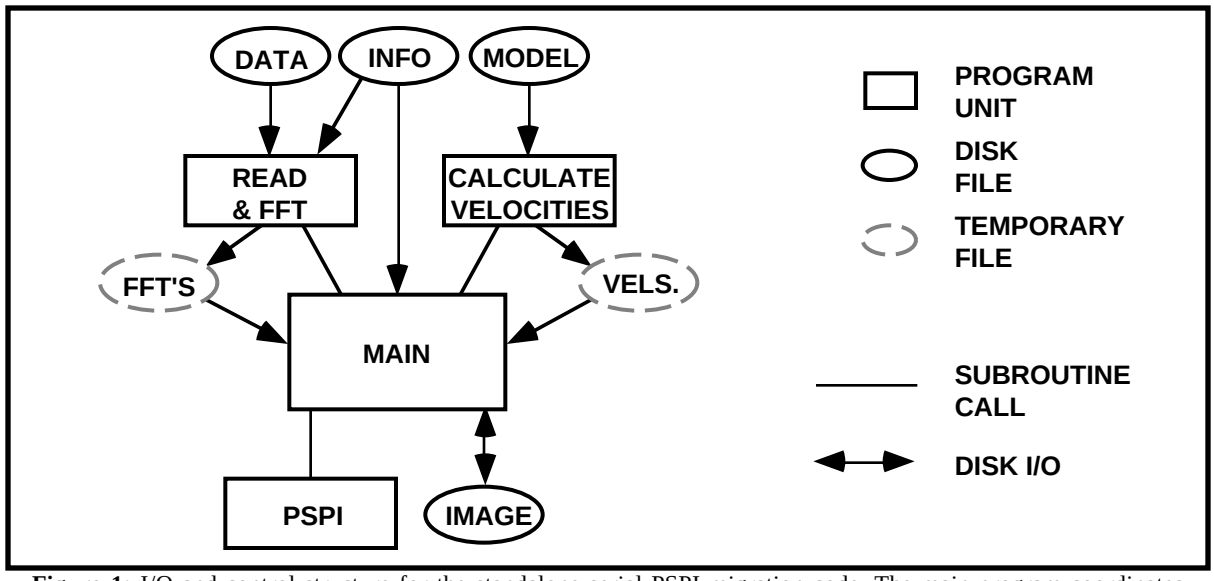

Figure 1: I/O and control structure for the standalone serial PSPI migration code. The main program coordinates all processing of records, which is done one at a time. Temporary scratch disk files are used to eliminate memory restrictions on the amount of data that can be processed at once.

Figure 2 shows the parallel control structure that was implemented on a heterogeneous cluster of Sun and SGI workstations. The master program is started up on one PE of the parallel virtual machine. It reads database information, creates an empty output image volume, spawns a slave migration process on each available PE, and sends a message to each slave telling them what record to begin processing. The slave process is similar to the main program of the serial code in that it still reads migration parameters and controls the reading of data, calculation of velocities, and PSPI migration. The main difference is that the migration code now operates blindly on whatever record the master tells it to process. When the slave has finished migrating a given record, it sends a message to the master and waits for a confirmation message that the master is ready to accept the results. After receiving the confirmation, the slave sends its results to the master in the form of a message and the master then updates the image volume and hands out another record number to the slave that just completed. The master constantly monitors the status and progress of each slave. If a slave crashes before its current record is completed, the master will detect this and hand out the partially migrated record to another slave for completion. After the last record has been handed out, the master will signal each slave to terminate when they have finished their current record.

The master/slave FIFO model described here was tested on a workstation cluster consisting of 6 Sun SPARC's, 2 SGI Indigo 2's and 2 SGI Indy's. With this 10-PE virtual machine, we found that initial message-passing and I/O bottlenecks occurred as multiple PE's were finishing their first record at approximately the same time. These bottlenecks disappeared after the first record because the slave processes became naturally staggered. Thus, we achieved predictable scaleability on the workstation cluster. Performance data for these runs is discussed below. Also, because the time required to update the migrated image is a fraction of the total time required for the migration itself, we estimated the staggering would be

sufficient to avoid bottlenecks for at least as many PE's as are available on the Cray T3D, which is 256. Although this turned out to be true in theory, there were other issues with the T3D machine architecture and native word size that limited the actual scaleability.

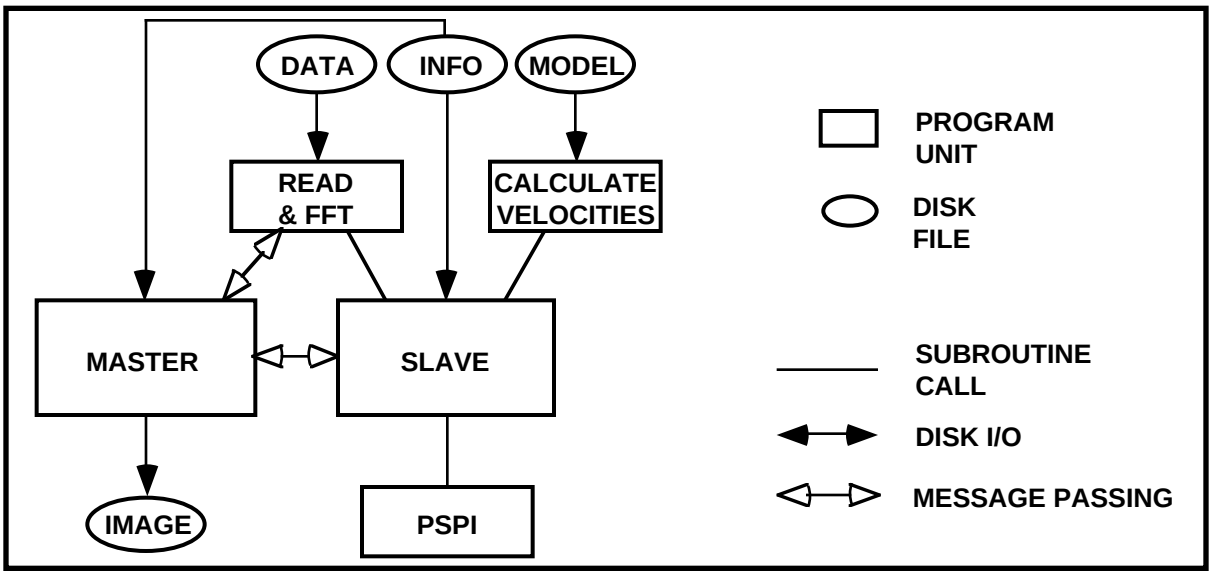

Figure 2: Master/slave control structure for the parallel implementation of the PSPI algorithm. The master process hands out record numbers to each slave process, receives migrated data and updates the output image volume. Scratch files were eliminated to avoid potential disk I/O bottlenecks on MPP machines.

Adaptation for Cray T3D

The two major issues that had to be addressed in porting this code from a workstation cluster to the T3D were that (1) I/O from the T3D processors and external peripherals is very slow compared to I/O from a workstation to its peripherals and (2) the physical memory available to each process was limited to 64 Mb, which is small compared to the several hundred Mb of virtual memory available on most workstations. Both issues were anticipated in the code development for the workstation environment to make the transition to the T3D as simple as possible.

The problem of slow I/O speed was addressed by storing all data in memory. This included both the intermediate data (velocity model and Fourier transformed shot data), stored on each slave machine, and the final image, stored on one or more master processors. Because the standard F77 compiler for the T3D only allowed 8-byte words, it was advantageous to use the newer F90 compiler instead, because it supports 4-byte words for both real and integer variables. This feature doubles the effective memory available to each PE and, thus, allowed the slaves to run more efficiently and halved the number of master PE's required to store the output image. Even so, for this particular problem, two master processes were required to store the non-zero part of the image volume.

We found the workstation and T3D implementations of PVM to be nearly identical. This made the porting of the communication sections of the code from the workstation environment to the T3D relatively easy. The major difference between the two PVM implementations was how the slave processes are started. In the workstation environment, a separate master process is started that probes the PVM virtual machine and spawns slave processes on all available PE's. On the T3D, one identical process is started on all allocated PE's. Each process determines whether it is a master or a slave, according to a processor number assigned to it by the operating system, and executes the appropriate section of the common code. One very nice feature of PVM is that the buffering and queuing of messages is handled entirely by the PVM routines. In fact, most of the message handling is done by the receiving routine on the master PE. This allows the slave processes to pass their parts of the image to the appropriate PVM routine and resume computing without waiting.

Performance Benchmark Results

Table 1 summarizes the benchmarking results obtained for the 10-PE workstation, 128-PE T3D and single-PE serial runs on 3 workstations with different CPU speeds. The test data set consisted of 265 common-shot records. Each record contained 768 traces and 751 samples per trace. The total size of the data set was approximately 330 Mb. The migrated image volume size was 240 X 300 X 400 (x,y,z). The single-PE runs completed in 130 to 260 hours, depending on the machine computing

speed. The 10-PE cluster was composed of a mixture of these 3 workstation types. The master process was run on the slowest machine and slave migration processes were spawned on the remaining 9 PE's. The 10-PE run completed the last shot record after 22 hours. This is roughly a factor of 9 faster than the time required for the SPARC serial run. Thus, performance scales directly with the number of slave PE's when compared with a serial run on a machine with the average CPU speed of all PE's in the virtual machine. Bottlenecks, as mentioned, will occur if the effective time to complete a single record becomes smaller than the time required for the master to receive the migrated data and update the image volume. For the 10-PE run, the effective processing time was 5 minutes per record. The time to update the image is approximately 30 seconds or less. Thus, bottlenecks should not occur until the number of PE's increases by another factor of 10 or more. The workstation code should behave well for virtual machines with at least 100 PE's. The T3D configuration used two master PE's and 126 slaves, and completed the last record after about 2 hours. This gives an effective processing speed of 30 seconds per record, which represents a 100-fold increase over the single-PE SPARC run. Thus, the scaleability on the T3D is somewhat limited, perhaps by the small size of the high-speed memory cache (8kb). Still, the performance increase is enough to make large migration runs feasible.

TABLE 1: Benchmark Performance Statistics

Input Data: 265 records X 768 traces X 751 samples Output Image: 240 X 300 X 400 (x,y,z)			
Number of PE's	Machine type	Time to migrate 265 records	Time per record
1	SGI Indy	260 hours	60 minutes
1	Sun SPARC	200 hours	45 minutes
1	SGI Indigo 2	130 hours	30 minutes
10	Sun/SGI cluster	22 hours	5 minutes
128	Cray T3D	2 hours	0.5 minutes

Conclusions

Certain algorithms for 3D prestack depth migration lend themselves naturally to parallel computing with simple message-passing models because they can operate on single common-shot or common-receiver records independently. The PSPI algorithm is a prime example of this. We have demonstrated how an existing serial computing version of the PSPI method can be easily ported to run in parallel on workstation clusters and MPP machines. The parallel implementation uses a simple master/slave model that requires very few modifications to the original serial code other than inserting the appropriate PVM message-passing calls and writing a master program to coordinate the multiple-record processing. We showed benchmark results comparing performance of the serial code with a 10-PE workstation run and a 128-PE Cray T3D run. Predictable scaleability was observed for these tests. In general, processing speed scaled in proportion to the total number of slave PE's and the average CPU speed of the cluster. The highest performance increase we observed, relative to a single-SPARC run, was approximately a factor of 100 and was limited, in theory, only by the number of available PE's and the time required to process a single record (machine speed). This is equivalent to reducing the total processing time for a typical large migration run from 3 months down to 1 day or less. The parallel model is generic enough that it can be easily used to port other migration algorithms that work on common-shot or common-receiver seismic data.

Acknowledgments

This research was performed in part using the resources located at the Advanced Computing Laboratory of Los Alamos National Laboratory, Los Alamos, NM 87545.

References

Gazdag, J. and P. Sguazzero: "Migration of Seismic Data by Phase Shift Plus Interpolation", *Geophysics*, **49,** No. 2, 124-131, 1984.

Yilmaz, O.: *Seismic Data Processing*, S.M. Doherty, ed., Society of Exploration Geophysicists, Tulsa, OK, 1987.